Shared Memory Parallelization

Thomas J. Watson Research Center
PO Box 218
Yorktown Heights, NY 10598

Outline

- What is Shared Memory Parallelization
- Variable Scoping
- Work Sharing
- Synchronization
- OpenMP
- Performance Issues
- Future of SMP

What is Shared Memory Parallelization

- All processors can access all the memory in the parallel system
- The time to access the memory may not be equal for all processors - not necessarily a flat memory
- Parallelizing on a SMP does not reduce CPU time - it reduces wallclock time - use rtc( )
- Parallel execution is achieved by generating threads which execute in parallel
- Number of threads is independent of the

What is Shared Memory Parallelization *(continued)*

- Overhead for SMP parallelization is large - size of parallel work construct must be significant enough to overcome overhead
- Runtime handling of parallel threads important
- SMP parallelization is degraded by other processes on the node - important to be dedicated on the SMP node
- Remember Amdahl's Law - Only get a speedup on code that is parallelized

Flat Profile - NAS Benchmark SP

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
27.4	31.51	31.51	40	787.75	787.75	.ztasweep [4]
22.5	57.39	25.88	41	631.22	631.22	.rhs [5]
20.6	81.06	23.67	40	591.75	591.75	.etasweep [6]
18.0	101.72	20.66	40	516.50	516.50	.xisweep [7]
10.2	113.45	11.73	1	11730.00	113450.00	.adi [3]
0.7	114.27	0.82	1	820.00	820.00	.erhs [8]
0.2	114.48	0.21	262904	0.00	0.00	.exact [10]
0.1	114.61	0.13	1	130.00	130.00	.setiv [11]
0.1	114.69	0.08	1	80.00	270.37	.error [9]
0.1	114.77	0.08	1	80.00	99.63	.setbv [12]
0.0	114.82	0.05				.__mcount [13]
0.0	114.83	0.01				._xlfReadLDInt [14]

Variable Scoping

- Most difficult part of Shared Memory Parallelization
  - What memory is Shared
  - What memory is Private - each processor has its own copy
- Fortran conception of Memory
  - Global
    - Shared by all routines
  - Local
    - Local to routine

Variable Scoping Rules

- Private Variables
  - A scalar variable that is set and then used within the DO is PRIVATE
  - An array whose subscript is constant with respect to the PARALLEL DO and is set and then used within the DO is PRIVATE
- Shared Variables
  - Everything Else
- SIMPLE - DIFFICULT to implement

Fortran vs SMP Scoping

- Whenever a Fortran GLOBAL variable is scoped PRIVATE or when a Fortran LOCAL variable is scoped SHARED problems arise
  - Variable passed into a routine scoped private - FIRST Value getting and LAST value setting
  - COMMON block variable within a called routine needs to be scoped private

OpenMP Directives

- http://www.openmp.org
- Comment line directives for
  - Scoping Data
  - Specifying Work Load
  - Synchronization of threads
- Function calls for obtaining information about threads

OpenMP Directives

- Scoping Variables
  - Default is shared
  - Can be set to NONE of PRIVATE
- Nothing like CRAY AUTOSCOPE - user responsible for scoping anything that is contrary to default
- Scoping cannot be done within a subroutine called from the parallel DO loop - except with THREADPRIVATE

OpenMP Directives

- !$OMP PARALLEL / !$OMP END PARALLEL
  - Indicate a parallel region for each thread to execute - must scope all variables within region

Default
Private
Shared
First
Private
Last
Private
Reduction
If

OpenMP Directives

- !$OMP PARALLEL DO / !$OMP END PARALLEL DO
  - Indicate a parallel do for all thread to shared in work - must scope all variables within region - Can specify Worksharing

Default
Private
Shared
First
Private
Last
Private
Reduction
If

SCHEDULE

OpenMP Directives

- !$OMP DO / !$OMP END DO
  - Indicate a parallel do for all thread to shared in work - May Scope variables Can specify Worksharing

Private
Shared
First
Private
Last
Private
Reduction

SCHEDULE

Variable Scoping

```
        read *,n
        sum = 0.0
        call random (b)
        call random (c)
!$OMP  PARALLEL DO
!$OMP&PRIVATE (i)
!$OMP&SHARED (a,b,n)
!$OMP&REDUCTION (+:sum)
       do i=1,n
        a(i) = sqrt(b(i)**2+c(i)**2)
        sum = sum + a(i)
       Enddo
!$OMP PARALLEL ENDDO
      end
```

Each processor needs
a separate copy of *i*
everything else is
Shared

Variable Scoping

```
      read *,n
      sum = 0.0
      call random (b)
      call random (c)
!$OMP PARALLEL
!$OMP PRIVATE (i,sump)
!$OMP SHARED (a,b,n,c,sum)
      sump = 0.0
!$OMP DO
     do i=1,n
      a(i) = sqrt(b(i)**2+c(i)**2)
      sump = sump + a(i)
     enddo
!$OMP CRITICAL
     sum = sum + sump
!$OMP ENDCRITICAL
!$OMP END PARALLEL
    end
```

Each processor needs

a separate copy of *i*

everything else is

Shared

Variable Scoping

```
          subroutine example4(n,m,a,b,c)
          real*8
a(100,100),B(100,100),c(100)
          integer n,i
          real*8 sum
!$OMP PARALLEL DO
!$OMP PRIVATE (j,i,c)
!$OMP SHARED (a,b,m,n)
           do j=1,m
             do i=2,n-1
              c(i) = sqrt(1.0+b(i,j)**2)
             enddo
             do i=1,n
              a(i,j) = sqrt(b(i,j)**2+c(i)**2)
             enddo
           enddo
           end
```

Each processor needs a separate copy of *j,i,c* everything else is Shared

What about *c*?
c(1) and *c(n)*?

Variable Scoping

```
      subroutine example4(n,m,a,b,c)
      real*8
a(100,100),B(100,100),c(100)
      integer n,i
      real*8 sum
!$OMP PARALLEL DO
!$OMP PRIVATE (j,i)
!$OMP SHARED (a,b,m,n)
!$OMP FIRSTPRIVATE (c)
       do j=1,m
        do i=2,n-1
         c(i) = sqrt(1.0+b(i,j)**2)
        enddo
        do i=1,n
         a(i,j) = sqrt(b(i,j)**2+c(i)**2)
        enddo
       enddo
       end
```

Need First Value of *c*

Master copies it's *c* array to all threads prior to DO loop

What about last value of *c* is it needed?

Variable Scoping

```
        subroutine example5(n,m,a,b,c)
        real*8a(100,100),B(100,100),c(100)
        real*8 cc(100)
        integer m,n,i
        real*8 sum
!$OMP PARALLEL
!$OMP PRIVATE (j,i,cc)
!$OMP SHARED (a,b,m,n)
        cc(1) = c(1)
        cc(n) = c(n)
!$OMP DO
        do j=1,m
         do i=2,n-1
           cc(i) = sqrt(1.0+b(i,j)**2)
         enddo
         do i=1,n
           a(i,j) = sqrt(b(i,j)**2+cc(i)**2)
         enddo
        enddo
!$OMP END DO
!$OMP END PARALLEL
        end
```

Need First Value of *c*

User copies what part of *c* is needed to all threads prior to DO loop

What about last value of *c* is it needed?

Variable Scoping

```
!$OMP PARALLEL
!$OMP PRIVATE (j,i)

!$OMP SHARED (a,b,m,n)
            cc(1) = c(1)
            cc(n) = c(n)
!$OMP DO
            do j=1,m
             do i=2,n-1
              cc(i) = sqrt(1.0+b(i,j)**2)
              enddo
              do i=1,n
               a(i,j) =
sqrt(b(i,j)**2+cc(i)**2)
               enddo
            enddo
!$OMP END DO
             if(j.eq.m+1)then
             do i=1,n
               c(i) = cc(i)
             enddo
             endif
!$OMP END PARALLEL
```

What about last value of *c* is it needed?

Calling an External from a Parallel Loop

```
       subroutine example5(n,m,a,b,c)
       real*8 a(100,100),B(100,100),c(100)
       integer m,n
!$OMP PARALLEL DO
!$OMP PRIVATE (j)
!$OMP SHARED (a,b,m,n)
        do j=1,m
         call doit(j,n,a,b)
       enddo
        end
```

```
subroutine doit(j,n,a,b)
real*8 a(100,100),B(100,100)
COMMON cc(100)
 do i=2,n-1
 IF(a(i,j).gt.SIN(b(i,j)))THEN
   cc(i) = sqrt(1.0+b(i,j)**2)
 ENDIF
 enddo
 do i=1,n
  a(i,j) = sqrt(b(i,j)**2+cc(i)**2)
 enddo
 end
```

Calling an External from a Parallel Loop

```
      subroutine example5(n,m,a,b,c)
      real*8 a(100,100),B(100,100),c(100)
      integer m,n
!$OMP PARALLEL DO
!$OMP PRIVATE (j)
!$OMP SHARED (a,b,m,n)
       do j=1,m
        call doit(j,n,a,b)
      enddo
       end
```

```
      subroutine doit(j,n,a,b)
      real*8 a(100,100),B(100,100)
!$OMP THREADPRIVATE (/BCOM/)
      COMMON/BCOM/ cc(100)
       do i=2,n-1
       IF(a(i,j).gt.SIN(b(i,j)))THEN
         cc(i) = sqrt(1.0+b(i,j)**2)
       ENDIF
       enddo
       do i=1,n
        a(i,j) = sqrt(b(i,j)**2+cc(i)**2)
       enddo
       end
```

Blank Common cannot appear on THREADPRIVATE

How about first value setting???

Not in xlf Version 6.1

Calling an External from a Parallel Loop

```
        subroutine example5(n,m,a,b,c)
        real*8 a(100,100),B(100,100),c(100)
!OMP$ THREADPRIVATE (/BCOM/)
        COMMON/BCOM/ cc(100)
        integer m,n
!OMP$ PARALLEL DO
!OMP$ PRIVATE (j)
!OMP$ SHARED (a,b,m,n)
!OMP$ PARALLEL DO COPYIN(/BCOM/)
         do j=1,m
          call doit(j,n,a,b)
        enddo
         end
```

```
        subroutine doit(j,n,a,b)
        real*8 a(100,100),B(100,100)
!OMP$ THREADPRIVATE (/BCOM/)
        COMMON/BCOM/ cc(100)
         do i=2,n-1
         IF(a(i,j).gt.SIN(b(i,j)))THEN
           cc(i) = sqrt(1.0+b(i,j)**2)
         ENDIF
         enddo
         do i=1,n
           a(i,j) = sqrt(b(i,j)**2+cc(i)**2)
         enddo
         end
```

Blank Common cannot appear on THREADPRIVATE
Entire Common Block need not be copied in

Not in xlf Version 6.1

Work Sharing Directives

- SCHEDULE (type,n)
  - Runtime
    - Scheduling is controlled by runtime environment variable
      - OMP_SCHEDULE
      - XLSMPOPTS on xlf Version 6.1

      Not in xlf Version 6.1
  - (Static,n)
    - Iterations are divided into chunks and pieces are statically assigned to threads in a round-robin fashion (Default n is iteration count/parthds)

Work Sharing Directives

- SCHEDULE
  - (Dynamic,n)
    - Work is divided into chunks of size n. As each thread finishes a chunk it dynamically obtains the next set of iterations. (default of n is 1)
  - (Guided,n)
    - Dynamic with chunksize starting at iterations/parthds, then exponentially decreasing to n. (default of n is 1)

Comparison of Work Sharing

Iterations	1000
Static,10	1-10, 41-50, 81-90 ... 11-20, 51-60, 91-100 ... 21-30, 61-70, 101-110 ... 31-40, 71-80, 111-120 ...
Dynamic,10	1-10, 71-80, 81-90 ... 11-20,91-100 ... 21-30,51-60, 61-70, 101-110 ... 31-40, 71-80, 111-120 ...
Guided	1-250, 686-764,927-945,971-978, ... 251-438,824-868,902-926 439-579,765-823,960-970 580-685,869-901,946-959

What Work Sharing for this?

```
          subroutine example5(n,m,a,b,c)
           real*8 a(100,100),B(100,100),c(100,100)
           integer i,j,m,n
!$OMP PARALLEL DO
!$OMP PRIVATE (i,j)
!$OMP SHARED (a,b,c,n,m)
          do i=1,m
            do j=i+1,n
              a(j,i) = sqrt(b(j,i)**2 + c(j,i)**2)
            enddo
           enddo
           end
```

QUIZ:

Whats wrong with this?

```
      subroutine example5(m,a,b,c)
       real*8 a(100),B(100),c(100)
       integer i,m
!$OMP PARALLEL DO
!$OMP PRIVATE (i)
!$OMP SHARED (a,b,c,m)
!$OMP SCHEDULE (static,10)
      do i=1,m
         a(i) = sqrt(b(i)**2 + c(i)**2)
        enddo
       end
```

Tradeoff Load Balancing and Reduced Overhead

- The larger the size (GRANULARITY) of the piece of work, the lower the overall thread overhead.
- The smaller the size (GRANULARITY) of the piece of work,the better the dynamically scheduled load balancing

OpenMP for C

- Specification 1.0, October 1998
- Same functionality as OpenMP for FORTRAN
- Differences in syntax:
  - #pragma omp parallel
  - #pragma omp for
- Differences in variable scoping:
  - variables "visible" when #pragma omp parallel encountered are shared by default
  - static variables declared within a parallel region are also shared
  - heap allocated memory (malloc) is shared (but pointer can be private)
  - automatic storage declared within a parallel region is private (ie, on the stack)

Invoking Parallelization on the Fortran Compile command

- xlf_r - Fortran 77
- xlf90_r - Fortran 90
- mpxlf_r - Fortran with MPI
  - -qsmp -qreport=smplist
    - recognizes OpenMP and does automatic parallelization
  - -qsmp=noauto
    - recognizes OpenMP and IBM and doesn't do automatic
  - -qsmp=omp

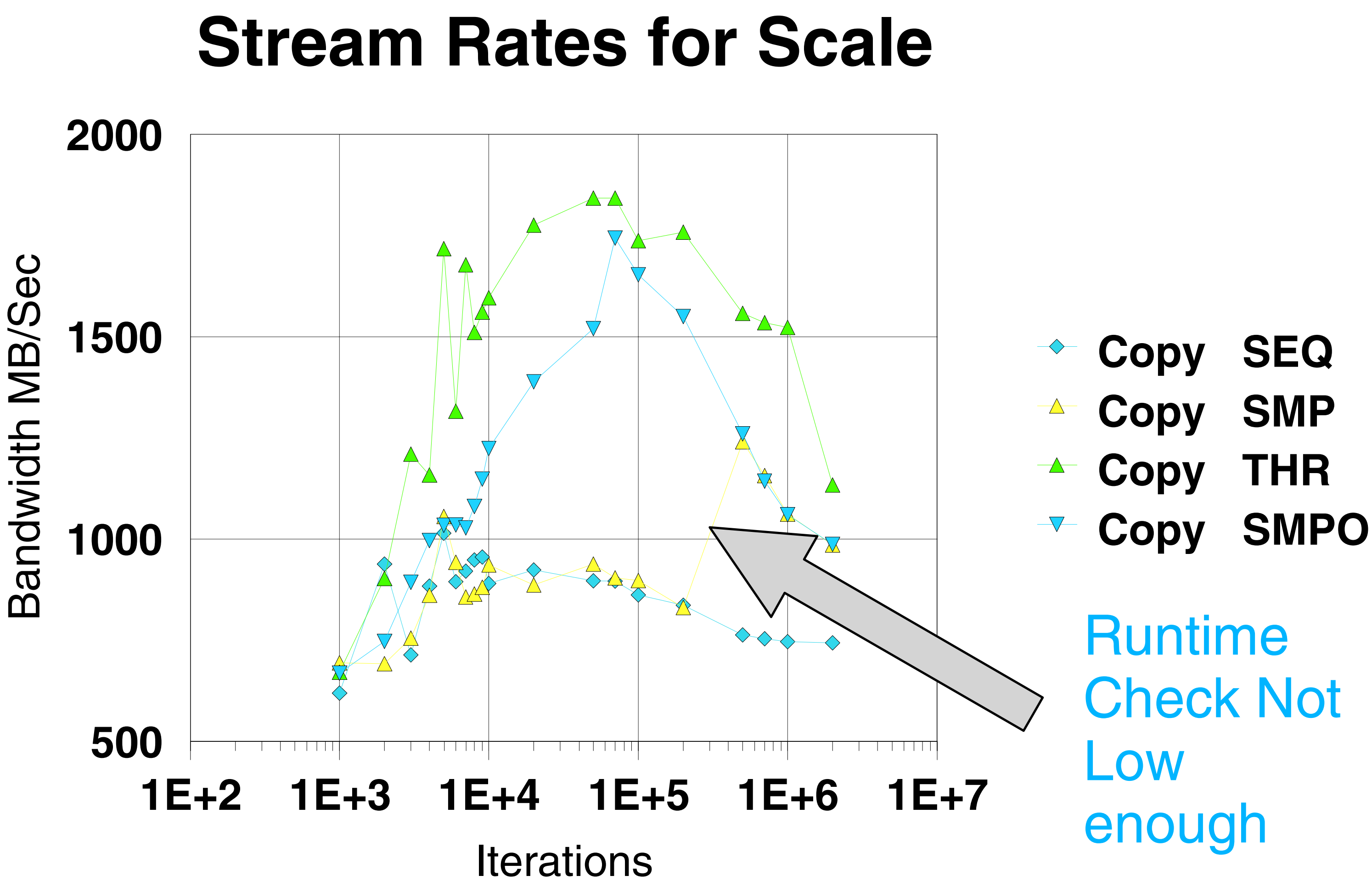
What About Automatic?

- xlf has a very good automatic parallelizer that might do a good job on a User's program.
  - When applying it across an entire program, some loops may slow down, some may speed up - you should be prepared to time individual loops before and after and then selectively parallelize what you want

What About Automatic?

- xlf has a very good automatic parallelizer that might do a good job on a User's program.
  - Runtime checking of overhead controlled by runtime environment variable

Winterhawk 2-Processor



Runtime Check Not Low enough

Runtime Enviroment Vars

- Some will probably change with Version 7.1 - OpenMP standard
  - XLSMPOPTS
    - parthreshold = num
      - specifies time in milliseconds below which the loop will run in serial
    - seqthreshold = num
      - specifies time beyond which previous sequential loop will be run in parallel
    - profilefreq=num
      - frequency with which loop should be analyzed
      - = 0 All profiling turned off

Runtime Management of Threads

- When the system encounters the first Parallelized DO loop the Master generates the worker threads and begins working on a chunk of the Parallel DO loop
- After the first Parallel DO loop is executed, all the worker threads are put to sleep - regardless of the spin Environment variable

Runtime Management of Threads *(continued)*

- When the next DO loop is encountered, the Master wakes up a first worker thread, the Master continues to work on the parallel loop. The first worker thread wakes up the second worker thread and starts to work on the parallel loop. The Master may have already started a second piece of work. The second thread wakes up the third, ...

Runtime Enviroment Vars

- Some will probably change with Version 7.1 - OpenMP standard
  - XLSMPOPTS
    - parthds=num
      - default - number of processors
    - stack=num
      - default - 4194304
    - spins=num (Only for locks)
      - default - 100
    - yields=num (Only for locks)
      - default - 10

Runtime Management of Threads

- Supply a runtime environment variable to specify that the threads should be put in a spin state rather than put to sleep.
- setenv SPINLOOPTIME 5000
  - This will saturate the CPU, not good if the node is timeshared
  - This will effectively reduce the overhead of threads joining the work section

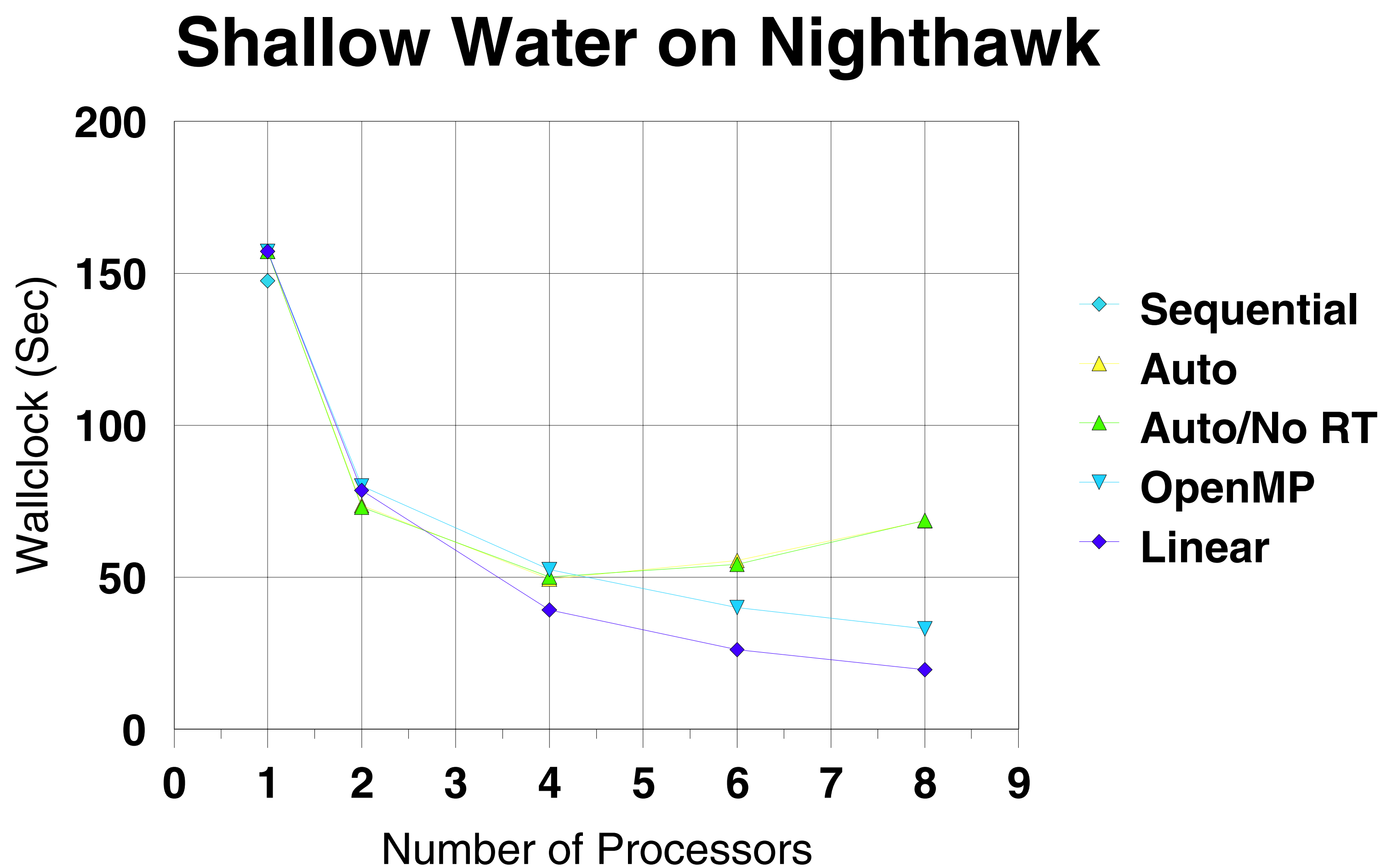
Some Real World Examples

- EMBAR
- SP
- BT
- MG
- SWIM

SWIM - SPEC 95

```
C$OMP PARALLELDO
C$OMP&SHARED (FSDY,FSDX,M,N,U,V,P,CU,CV,Z,H)
C$OMP&PRIVATE (I,J)
      DO 100 j = 1, n
        DO 100 i = 1, m
          cu(i + 1, j) = .5 * (p(i + 1, j) + p(i, j)) * u(i + 1, j)
          cv(i, j + 1) = .5 * (p(i, j + 1) + p(i, j)) * v(i, j + 1)
          z(i + 1, j + 1) = (fsdx * (v(i + 1, j + 1) - v(i, j + 1)) -
     .        fsdy * (u(i + 1, j + 1) - u(i + 1, j))) / (p(i, j)
     .        + p(i + 1, j) + p(i + 1, j + 1) + p(i, j + 1))
          h(i, j) = p(i, j) + .25 * (u(i + 1, j) * u(i + 1, j) + u(i, j)
     .        *u(i, j) + v(i, j + 1) * v(i, j + 1) + v(i, j) * v(i, j))
  100 CONTINUE
```

Shallow Water on Nighthawk



NAS Benchmarks